Utility-based Optimal Service Selection for Business Processes in Service Oriented Architectures

Vinod K. Dubey, Daniel A. Menascé


Presented by David González
Overview

- Objectives and Concluding Remarks
- QoS composition, cost and utility functions
- JOSeS vs HCB
- Experiment
- Results
- Discussion
Objectives and Concluding Remarks

- Address optimal service selection problem for business processes in SOA environments.
- Provided a optimal solution, Extended JOSeS, and a heuristic solution, HCB.
- The heuristic solution performed 99.5% close to the optimal solution using significantly less points from the solution space and computing resources.

- Now let’s see how to get there.
Optimization Problem

Maximize \( U(E[R(z)], A(z), X(z)) \)

subject to

\[ E[R(z)] \leq R_{\text{max}} \]
\[ A_{\text{min}} \leq A(z) \leq 1 \]
\[ X(z) \geq X_{\text{min}} \]
\[ C(z) \leq C_{\text{max}} \]
\[ z \in Z \]
Assumptions and BPEL

- Availability and Throughput are deterministic.
- End-to-end execution time and cost are nondeterministic.

Figure 1: An example of a BPEL business process on the left, the corresponding BPTree on the right, and an execution graph on the middle.
Computing Availability and Throughput

$q$ is the probability that activity $a_i$ is invoked [10].
Computing end-to-end execution time

The expected value of a maximum of a set of independent random variables[10]
Utility functions

\[ U_i(v(z)) = K_i \frac{e^{\alpha_i(\beta_i - v(z))}}{1 + e^{\alpha_i(\beta_i - v(z))}} \]  \hspace{1cm} (2)

\[ U_g(z) = \left( \prod_{i=1}^{3} (U_i(z))^w_i \right) \sum_{j=1}^{3} w_j \]  \hspace{1cm} (3)
JOSeS vs HCB

- Jensen-based Optimal Service Selection (JOSeS). This algorithm does not require one to generate the entire solution space $Z$, but only a subset of the solution space where each point represents a feasible solution.

- Hill-Climbing Based (HCB), which defines a neighborhood of the point currently being visited and move to the best point in the neighborhood. The process continues until a near-optimum solution is found given a stopping criterion.
Optimal Solution: JOSeS

Algorithm 3 AdvanceList Function

```java
public function AdvanceList (k) returns (s)
1: s ← next (k);
2: if s = NULL then
3:     if k > 1 then
4:         reset (k); k ← k - 1; z ← z;
5:         AdvanceList (k);
6:     else
7:         return s
8: end if
9: else
10:     return s;
11: end if
```

Algorithm 4 JOSeS Algorithm to Compute the Optimal Service Selection Optimizing the Global Utility

```java
function OptimalSolution() returns (z)
1: reset (1); k ← 1; /* initialize activity pointers */
2: s ← AdvanceList (k); z ← s; /* initialize solution */
3: zopt ← any allocation in Z;
4: while s ≠ NULL do
5:     if k < N then
6:         if (E[R(z)] ≤ Rmax) ∧ (A(z) ≥ Amin) ∧ (X(z) ≥ Xmin) ∧ (C(z) ≤ Cmax) then
7:             k ← k + 1
8:         else
9:             z ← z; /* remove last SP in z */
10:            end if
11:     else
12:         if (E[R(z)] ≤ Rmax) ∧ (A(z) ≥ Amin) ∧ (X(z) ≥ Xmin) ∧ (C(z) ≤ Cmax) then
13:             if U(z) > U(zopt) then
14:                 zopt ← z
15:             end if
16:         end if
17:     else
18:         z ← z; /* remove last SP in z */
19:     end if
20:     s ← AdvanceList (k); z ← z||s
21: end while
22: return zopt
23: end function
```
Heuristic Solution: HCB(1)

Algorithm 6 Identify Neighbors

1: function neighbors (z₀) returns (Z)
2: Z ← ∅; /* Initialize with empty neighborhood */
3: N ← ∅; /* All neighbors */
4: for all activity i = 1, ..., N do
5:     for all q_i ∈ {R_i, A_i, X_i} do
6:         if q_i = R_i then
7:             /* s = best improvement in response time */
8:             s = arg max_{k=1}^{S_i} \left\{ 1 - \frac{q_{i,k}}{q_{i,curr}} \right\};
9:         else
10:             /* s = best improvement in availability and throughput */
11:             s = arg max_{k=1}^{S_i} \left\{ \frac{q_{i,k}}{q_{i,curr}} - 1 \right\};
12:         end if
13:     z = replace (z₀, i, s); /* Replace current SP of a_i in z₀ by s */
14:     if z ∉ N then
15:         N ← N ∪ z;
16:         if (((C(z) ≤ C_{max}) and (A(z) ≥ A_{min}) and
17:            (L(E[R(z)]) ≤ R_{max}) and (X(z) ≥ X_{min}) \))
18:             then
19:                 if (E[R(z)] ≤ R_{max}) then
20:                     Z ← Z ∪ z;
21:             end if
22:         end if
23:     end for
24: end for
25: return Z;
Algorithm 5 HCB Heuristic Algorithm

1: function HeuristicSolution() returns (z)
2: nrestarts ← 0;
3: while (nrestarts < maxrestarts) do
4:     $z_0$ ← randomStart(); /* random start */
5:     nrestarts ← nrestarts + 1; searching ← TRUE;
6:     while (searching) do
7:         $\mathcal{Z}$ ← neighbors ($z_0$); /* get feasible neighbors */
8:         $z_{opt} ← \arg\max_{z \in \mathcal{Z}} U(z)$; /* Identify neighbor with highest utility */
9:         if ($U(z_{opt}) > U(z_0)$) then
10:             $z_0 ← z_{opt}$;
11:         else
12:             searching ← FALSE; /* local optimum */
13:         end if
14:     end while
15:     if (nrestarts = 1) then
16:         $z_{gopt} ← z_{opt}$;
17:     else if ($U(z_{opt}) > U(z_{gopt})$) then
18:         $z_{gopt} ← z_{opt}$;
19:     end if
20: end while
21: return $z_{gopt}$;
22: end function
Experiment

- Aimed to evaluate the efficiency between the algorithms; solution space required and computation time by them; and compare them based on other parameters such complexity of the BPT and SPs per activity.

- 50 BPEL business processes were randomly generated, which contained 6 to 9 activities and had constructs such as sequence, flow, and switch-case. A total of 36000 runs were made.

- The calculations were made using a 95% confidence interval.
Results: Utilization ratio comparison

Figure 1. Average $U_h/U_o$ (%) vs. n_spa for four constraint strengths

Figure 4. Average $U_h/U_o$ (%) vs. n_spa for simple, medium, and complex business processes
Results: Number of points examined comparison

Figure 2. Average number of points examined $N_h$ and $N_o$ vs. $n_{spa}$ for four constraint strengths

Figure 5. Average number of points examined $N_h$ and $N_o$ vs. $n_{spa}$ for simple, medium, and complex business processes
Results: Computing time comparison

Figure 3. Average computation time $T_h$ and $T_o$ vs. $n_{spa}$ for four constraint strengths
Results: Analysis of the Nh visited points growth against SPs per activity

\[ Nh = 39 \times \text{NSPA} \]
\[ R^2 = 0.99678 \]

Figure 6. Average \( N_h \) vs. \( n_{spa} \)
Discussion

- Has HCB solution runtime limitations?
- What is next step after HCB?

Thank you for your time!